```
# Since delivery details of one package are divided into several rows (think of it as connect
#  Now think about how we should treat their fields if we combine these rows? What aggregatio
#  What would happen to the numeric fields if we merge the rows?

# Hint: You can use inbuilt functions like groupby and aggregations like sum(), cumsum() to m
# 1. Trip_uuid, Source ID and Destination ID 2. Further aggregate on the basis of just Trip_u
# You can also keep the first and last values for some numeric/categorical fields if aggregat

# Basic data cleaning and exploration:
# Handle missing values in the data.
# Analyze the structure of the data.
# Try merging the rows using the hint mentioned above.
# Build some features to prepare the data for actual analysis. Extract features from the belo
# Destination Name: Split and extract features out of destination. City-place-code (State)
# Source Name: Split and extract features out of destination. City-place-code (State)
# Trip_creation_time: Extract features like month, year and day etc
# In-depth analysis and feature engineering:

# Calculate the time taken between od_start_time and od_end_time and keep it as a feature. Dr

# Compare the difference between Point a. and start_scan_to_end_scan. Do hypothesis testing/ 

# Do hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time a
# (aggregated values are the values you'll get after merging the rows on the basis of trip_uu

# Do hypothesis testing/ visual analysis between actual_time aggregated value and segment act
# (aggregated values are the values you'll get after merging the rows on the basis of trip_uu

# Do hypothesis testing/ visual analysis between osrm distance aggregated value and segment o
# (aggregated values are the values you'll get after merging the rows on the basis of trip_uu

# Do hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm 
# (aggregated values are the values you'll get after merging the rows on the basis of trip_uu

# Find outliers in the numerical variables (you might find outliers in almost all the variabl

# Handle the outliers using the IQR method.

# Do one-hot encoding of categorical variables (like route_type)

# Normalize/ Standardize the numerical features using MinMaxScaler or StandardScaler.



# Evaluation Criteria (100 Points):

# Define Problem Statement and perform Exploratory Data Analysis (10 points)
# Definition of problem (as per given problem statement with additional views)
# Observations on shape of data, data types of all the attributes, conversion of categorical
#  missing value detection, statistical summary.
# Visual Analysis (distribution plots of all the continuous variable(s), boxplots of all the
```

```
# Visual Analysis (distribution plots of all the continuous variable(s), boxplots of all the
# Insights based on EDA
# Comments on range of attributes, outliers of various attributes
# Comments on the distribution of the variables and relationship between them
# Comments for each univariate and bivariate plot


# Feature Creation (10 Points)
# Merging of rows and aggregation of fields (10 Points)
# Comparison & Visualization of time and distance fields (10 Points)
# Missing values Treatment & Outlier treatment (10 Points)
# Checking relationship between aggregated fields (10 Points)
# Handling categorical values (10 Points)
# Column Normalization /Column Standardization (10 Points)
# Business Insights (10 Points) - Should include patterns observed in the data along with wha
# Check from where most orders are coming from (State, Corridor etc)
# Busiest corridor, avg distance between them, avg time taken
# Recommendations (10 Points) - Actionable items for business. No technical jargon. No compli
```

```
import numpy as np, pandas as pd,seaborn as sns,matplotlib.pyplot as plt
```

```
from scipy.stats import kruskal,pearsonr,chi2_contingency
```

```
!gdown "https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/delhi
```

```
Downloading...
From: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/551/original/de
To: /content/delhivery_data.csv?1642751181
100% 55.6M/55.6M [00:04<00:00, 11.2MB/s]
```

```
df = pd.read_csv("/content/delhivery_data.csv?1642751181")
```

```
df.head()
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	so
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IN
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IN
2	training	2018-09-20	thanos::sroute:eb7bfc78-	Carting	trip-	IN

```
df.columns
```

```
Index(['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor'],
      dtype='object')
```

```
df.shape
```

```
(144867, 24)
```

The DataFrame consists of 24 columns with 144867 records.

```
df.dtypes
```

```
data                               object
trip_creation_time                 object
route_schedule_uuid                object
route_type                         object
trip_uuid                          object
source_center                      object
source_name                        object
destination_center                 object
destination_name                   object
od_start_time                      object
od_end_time                        object
start_scan_to_end_scan            float64
is_cutoff                            bool
cutoff_factor                       int64
cutoff_timestamp                   object
actual_distance_to_destination    float64
actual_time                       float64
osrm_time                         float64
osrm_distance                     float64
factor                            float64
segment_actual_time               float64
```

```
segment_osrm_time               float64
segment_osrm_distance           float64
segment_factor                  float64
dtype: object
```

As seen in dataframe and dataframes data types, all columns those datatypes should be Date-Time but have object instead. So need to change the datatypes of columns.

```
cols = ['trip_creation_time','od_start_time','od_end_time','cutoff_timestamp']
for i in cols:
  df[i]=pd.to_datetime(df[i])
```

```
df.dtypes
```

```
data                                      object
trip_creation_time                datetime64[ns]
route_schedule_uuid                       object
route_type                                object
trip_uuid                                 object
source_center                             object
source_name                               object
destination_center                        object
destination_name                          object
od_start_time                     datetime64[ns]
od_end_time                       datetime64[ns]
start_scan_to_end_scan                   float64
is_cutoff                                   bool
cutoff_factor                              int64
cutoff_timestamp                  datetime64[ns]
actual_distance_to_destination           float64
actual_time                              float64
osrm_time                                float64
osrm_distance                            float64
factor                                   float64
segment_actual_time                      float64
segment_osrm_time                        float64
segment_osrm_distance                    float64
segment_factor                           float64
dtype: object
```

```
df.head(3)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	so
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IN
		2018-09-20	thanos::sroute:eb7bfc78-		trip-	

```
df.isnull().sum()
```

```
data                               0
trip_creation_time                 0
route_schedule_uuid                0
route_type                         0
trip_uuid                          0
source_center                      0
source_name                      293
destination_center                 0
destination_name                 261
od_start_time                      0
od_end_time                        0
start_scan_to_end_scan             0
is_cutoff                          0
cutoff_factor                      0
cutoff_timestamp                   0
actual_distance_to_destination     0
actual_time                        0
osrm_time                          0
osrm_distance                      0
factor                             0
segment_actual_time                0
segment_osrm_time                  0
segment_osrm_distance              0
segment_factor                     0
dtype: int64
```

As the Null values present in the dataframe are very less significantly equals to 0.2%, so by dropping them wont affect the dataframe.

```
df.dropna(inplace=True)
```

```
df.isnull().sum()
```

```
data                           0
trip_creation_time             0
route_schedule_uuid            0
route_type                     0
trip_uuid                      0
source_center                  0
source_name                    0
destination_center             0
destination_name               0
```

```
od_start_time                     0
od_end_time                       0
start_scan_to_end_scan            0
is_cutoff                         0
cutoff_factor                     0
cutoff_timestamp                  0
actual_distance_to_destination    0
actual_time                       0
osrm_time                         0
osrm_distance                     0
factor                            0
segment_actual_time               0
segment_osrm_time                 0
segment_osrm_distance             0
segment_factor                    0
dtype: int64
```

We can see that there are no null values.

```
# Checking which columns are Categorical and which are Numerical:

data = ['data', 'trip_creation_time', 'route_schedule_uuid', 'route_type',
       'trip_uuid', 'source_center', 'source_name', 'destination_center',
       'destination_name', 'od_start_time', 'od_end_time',
       'start_scan_to_end_scan', 'is_cutoff', 'cutoff_factor',
       'cutoff_timestamp', 'actual_distance_to_destination', 'actual_time',
       'osrm_time', 'osrm_distance', 'factor', 'segment_actual_time',
       'segment_osrm_time', 'segment_osrm_distance', 'segment_factor']
lis = []
for i in data:
  print(i, " : ",df[i].nunique())
```

```
data  :  2
trip_creation_time  :  14787
route_schedule_uuid  :  1497
route_type  :  2
trip_uuid  :  14787
source_center  :  1496
source_name  :  1496
destination_center  :  1466
destination_name  :  1466
od_start_time  :  26223
od_end_time  :  26223
start_scan_to_end_scan  :  1914
is_cutoff  :  2
cutoff_factor  :  501
cutoff_timestamp  :  92894
actual_distance_to_destination  :  143965
actual_time  :  3182
osrm_time  :  1531
osrm_distance  :  137544
factor  :  45588
segment_actual_time  :  746
```

```
segment_osrm_time  :  214
segment_osrm_distance  :  113497
segment_factor  :  5663
```

We can observe that there are 3 Categorical Columns seen from the dataframe and that are:

1. Data
2. Route_type
3. is_cutoff

```
df.columns.value_counts(normalize=True)*100
```

```
data                              4.166667
trip_creation_time                4.166667
segment_osrm_distance             4.166667
segment_osrm_time                 4.166667
segment_actual_time               4.166667
factor                            4.166667
osrm_distance                     4.166667
osrm_time                         4.166667
actual_time                       4.166667
actual_distance_to_destination    4.166667
cutoff_timestamp                  4.166667
cutoff_factor                     4.166667
is_cutoff                         4.166667
start_scan_to_end_scan            4.166667
od_end_time                       4.166667
od_start_time                     4.166667
destination_name                  4.166667
destination_center                4.166667
source_name                       4.166667
source_center                     4.166667
trip_uuid                         4.166667
route_type                        4.166667
route_schedule_uuid               4.166667
segment_factor                    4.166667
dtype: float64
```

```
df['data'].value_counts()
```

```
training    104632
test         39684
Name: data, dtype: int64
```

```
print("Route_type: ",df['route_type'].value_counts())
print("Cutoff: ",df['is_cutoff'].value_counts())
```

```
Route_type:  FTL        99132
Carting    45184
Name: route_type, dtype: int64
```

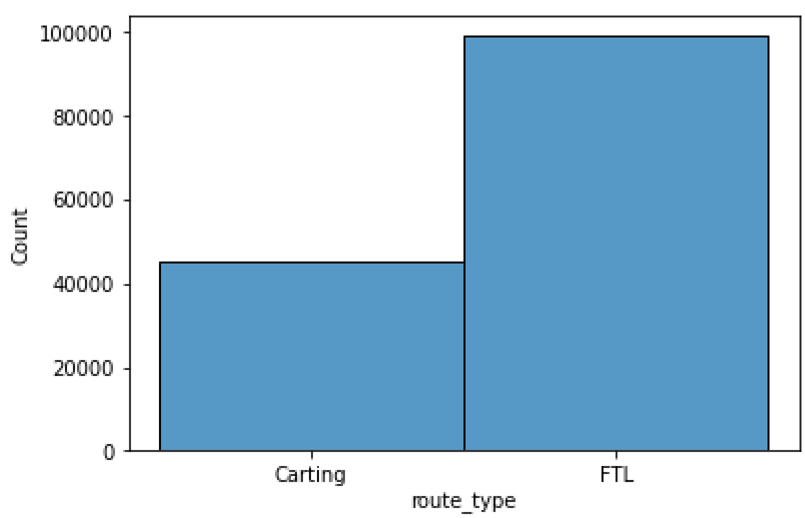
```
Cutoff:  True     118336
False     25980
Name: is_cutoff, dtype: int64
```

```
df.describe()
```

	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_
count	144316.000000	144316.000000	144316.000000	144316.00
mean	963.697698	233.561345	234.708498	417.99
std	1038.082976	345.245823	345.480571	598.94
min	20.000000	9.000000	9.000045	9.00
25%	161.000000	22.000000	23.352027	51.00
50%	451.000000	66.000000	66.135322	132.00
75%	1645.000000	286.000000	286.919294	516.00
max	7898.000000	1927.000000	1927.447705	4532.00

Uni-Variate Analysis

```
sns.histplot(x=df['route_type'])
plt.show()
```




```
min(df['od_start_time'])
```

```
Timestamp('2018-09-12 00:00:16.535741')
```

```
max(df['od_end_time'])
```

```
Timestamp('2018-10-08 03:00:24.353479')
```

DataFrame timestamps are between 12-Sept-2018 to 08-Oct-2018.

```
# Calculate the time taken between od_start_time and od_end_time and keep it as a feature. Dr
df['total_time_in_od'] = (df['od_end_time']-df['od_start_time'])
df.drop(['od_end_time','od_start_time'],axis=1)
```

```
# cols1 = ['start_scan_to_end_scan']
# for i in cols1:
#   df[i]=pd.to_datetime(df[i])
```

```
cols = ['od_start_time','od_end_time']
for i in cols:
  df[i]=pd.to_datetime(df[i])
```

```
# Compare the difference between Point a. and start_scan_to_end_scan. Do hypothesis testing/
```

```
# x=df['total_time_in_od']-df['start_scan_to_end_scan']
```

```
df.dtypes
```

```
data                                      object
trip_creation_time                datetime64[ns]
route_schedule_uuid                       object
route_type                                object
trip_uuid                                 object
source_center                             object
source_name                               object
destination_center                        object
destination_name                          object
od_start_time                     datetime64[ns]
od_end_time                       datetime64[ns]
start_scan_to_end_scan                   float64
is_cutoff                                   bool
cutoff_factor                              int64
cutoff_timestamp                  datetime64[ns]
actual_distance_to_destination           float64
actual_time                              float64
osrm_time                                float64
osrm_distance                            float64
factor                                   float64
segment_actual_time                      float64
segment_osrm_time                        float64
segment_osrm_distance                    float64
segment_factor                           float64
```

```
    total_time_in_od                  timedelta64[ns]
    dtype: object
```

```
secs = []
for i in new_df_cap['total_time_in_od']:
  secs.append((i.total_seconds())/3600)
```

```
new_df_cap.drop(['new_col'],axis=1)
```

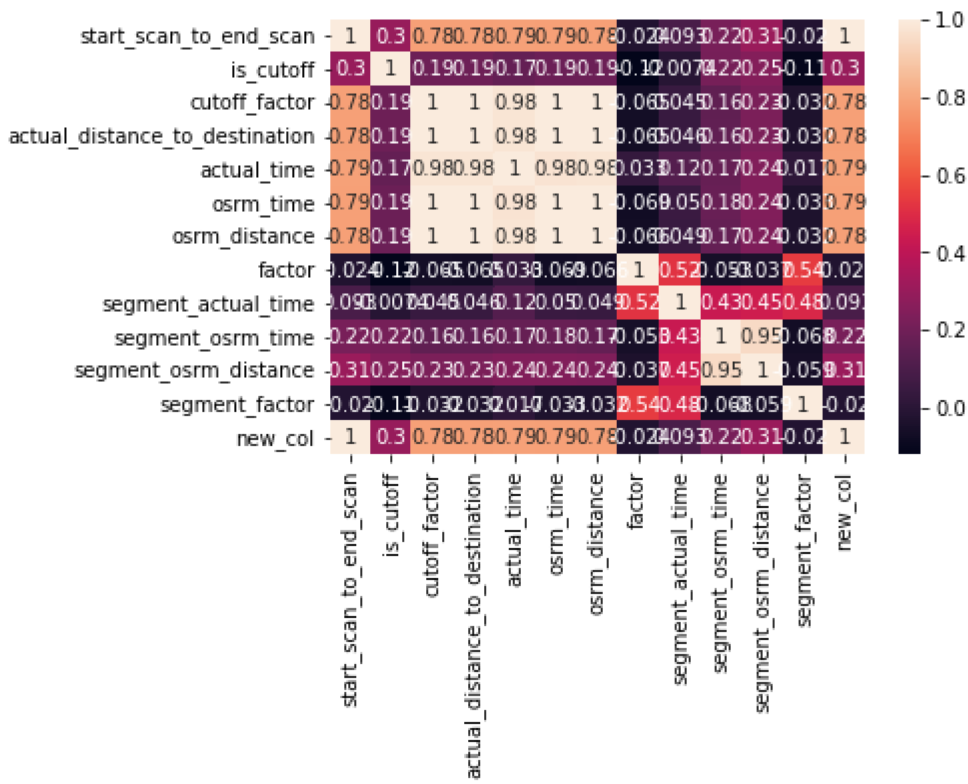
```
new_df_cap['new_col'] = np.array(secs)
```

```
new_df_cap.head(2)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	so
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IN
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IN

2 rows × 33 columns

```
sns.heatmap(df.corr(),annot=True)
plt.show()
```



We might see that there are few correlations between columns:

1.cutoff_factor - [actual_distance_to_destination,actual_time,osrm_time,osrm_distance].

2.segment_osrm_distance - segment_osrm_time

3.time_difference_between_od_timings - start_scan_to_end_scan

```
fig,axes = plt.subplots(nrows = 2,ncols = 3,figsize=(10,15))
sns.boxplot(data = df,x='cutoff_factor',ax = axes[0,0])
sns.boxplot(data = df,x='actual_time',ax = axes[1,0])
sns.boxplot(data = df,x='segment_actual_time',ax = axes[0,1])
sns.boxplot(data = df,x='segment_osrm_time',ax = axes[1,1])
sns.boxplot(data = df,x='new_col',ax = axes[0,2])
sns.boxplot(data = df,x='start_scan_to_end_scan',ax = axes[1,2])
plt.show()
```

Handling Outliers from each columns:

1. Cutoff-Factor
2. actual_time
3. Segment_orsm_time
4. start_scan_to_end_scan
5. segment_actual_time

```
# Cutoff_Factor outliers handle:
cut_off_25 = df['cutoff_factor'].quantile(0.25)
cut_off_75 = df['cutoff_factor'].quantile(0.75)
iqr = cut_off_75 - cut_off_25
upper_limit = cut_off_75 + 1.5 * iqr
lower_limit = cut_off_25 - 1.5 * iqr


new_df=df[df['cutoff_factor'] > upper_limit]
new_df.shape
```

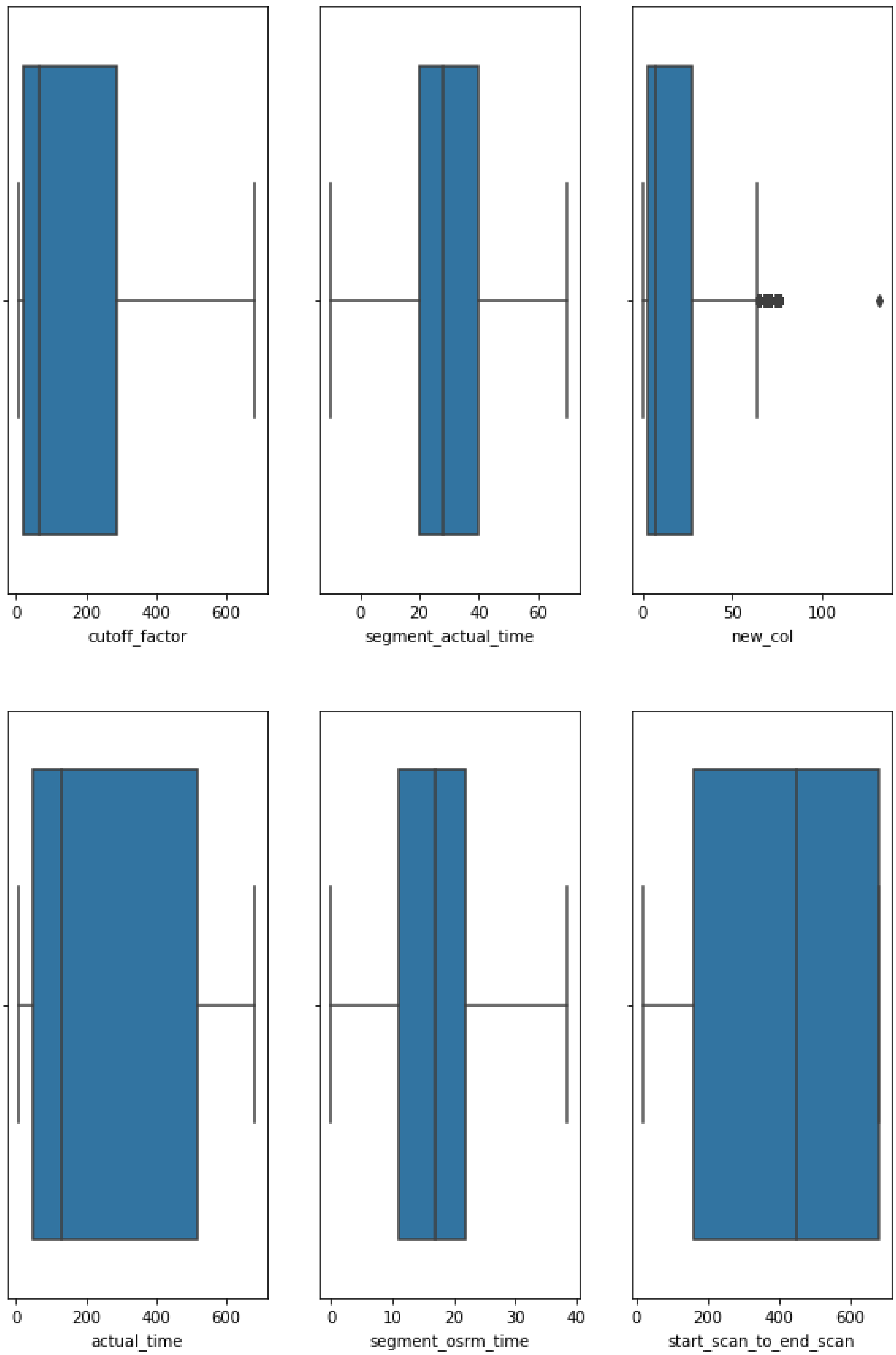
```
(17246, 33)
```

```
new_df_cap = df.copy()

new_df_cap['cutoff_factor'] = np.where(
    new_df_cap['cutoff_factor'] > upper_limit,
    upper_limit,
    np.where(
        new_df_cap['cutoff_factor'] < lower_limit,
        lower_limit,
        new_df_cap['cutoff_factor']
    )
)
```

```
segment_actual_time_25 = df['segment_actual_time'].quantile(0.25)
segment_actual_time_75 = df['segment_actual_time'].quantile(0.75)
iqr2 = segment_actual_time_75 - segment_actual_time_25
upper_limit2= segment_actual_time_75 + 1.5 * iqr2
lower_limit2 = segment_actual_time_25 - 1.5 * iqr2

new_df_cap['segment_actual_time'] = np.where(
    new_df_cap['segment_actual_time'] > upper_limit2,
    upper_limit2,
```

```
    np.where(
        new_df_cap['segment_actual_time'] < lower_limit2,
        lower_limit2,
        new_df_cap['segment_actual_time']
    )
)

start_scan_to_end_scan_25 = df['start_scan_to_end_scan'].quantile(0.25)
start_scan_to_end_scan_75 = df['start_scan_to_end_scan'].quantile(0.75)
iqr3 = start_scan_to_end_scan_75 - start_scan_to_end_scan_25
upper_limit3= start_scan_to_end_scan_75 + 1.5 * iqr3
lower_limit3 = start_scan_to_end_scan_25 - 1.5 * iqr3

new_df_cap['start_scan_to_end_scan'] = np.where(
    new_df_cap['start_scan_to_end_scan'] > upper_limit3,
    upper_limit3,
    np.where(
        new_df_cap['start_scan_to_end_scan'] < lower_limit3,
        lower_limit3,
        new_df_cap['start_scan_to_end_scan']
    )
)

Segment_orsm_time_25 = df['segment_osrm_time'].quantile(0.25)
Segment_orsm_time_75 = df['segment_osrm_time'].quantile(0.75)
iqr4 = Segment_orsm_time_75 - Segment_orsm_time_25
upper_limit4= Segment_orsm_time_75 + 1.5 * iqr4
lower_limit4 = Segment_orsm_time_25 - 1.5 * iqr4

new_df_cap['segment_osrm_time'] = np.where(
    new_df_cap['segment_osrm_time'] > upper_limit4,
    upper_limit4,
    np.where(
        new_df_cap['segment_osrm_time'] < lower_limit4,
        lower_limit4,
        new_df_cap['segment_osrm_time']
    )
)
```

```
fig,axes = plt.subplots(nrows = 2,ncols = 3,figsize=(10,15))
sns.boxplot(data = new_df_cap,x='cutoff_factor',ax = axes[0,0])
sns.boxplot(data = new_df_cap,x='actual_time',ax = axes[1,0])
sns.boxplot(data = new_df_cap,x='segment_actual_time',ax = axes[0,1])
sns.boxplot(data = new_df_cap,x='segment_osrm_time',ax = axes[1,1])
sns.boxplot(data = new_df_cap,x='new_col',ax = axes[0,2])
sns.boxplot(data = new_df_cap,x='start_scan_to_end_scan',ax = axes[1,2])
plt.show()
```



In all columns most of the Outliers were controlled in space of IQR.

Checking effect on Data wrt outliers.

```
df.describe()
```

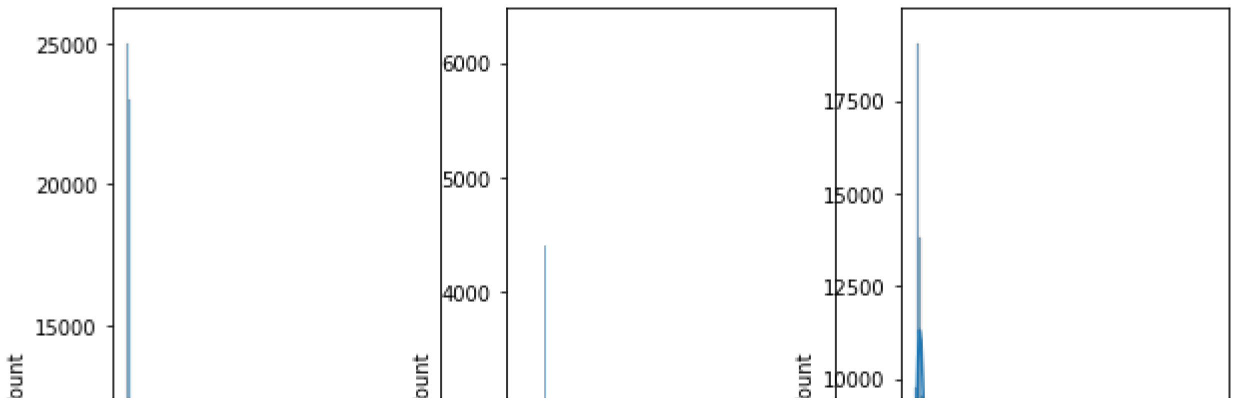
	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_
count	144316.000000	144316.000000	144316.000000	144316.00
mean	963.697698	233.561345	234.708498	417.99
std	1038.082976	345.245823	345.480571	598.94
min	20.000000	9.000000	9.000045	9.00
25%	161.000000	22.000000	23.352027	51.00
50%	451.000000	66.000000	66.135322	132.00
75%	1645.000000	286.000000	286.919294	516.00
max	7898.000000	1927.000000	1927.447705	4532.00

```
new_df_cap.describe()
```

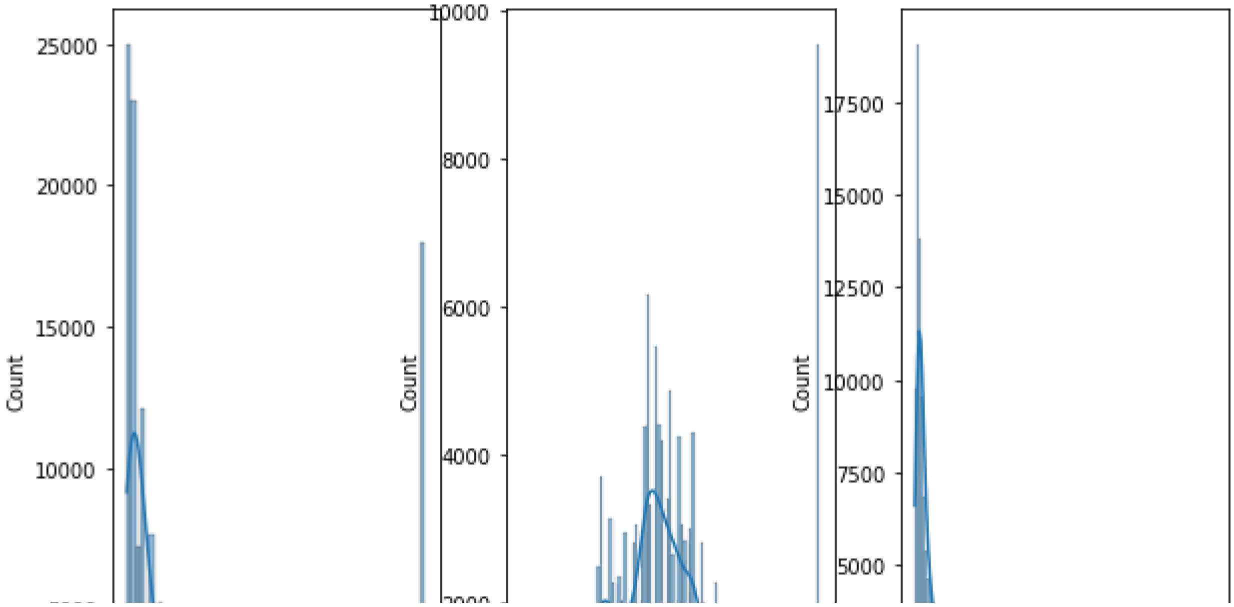
	start_scan_to_end_scan	cutoff_factor	actual_distance_to_destination	actual_
mean	423.558873	190.763852	234.708498	263.26

If we compare the data in both dataframes i.e without outliers and with outliers, we can see huge amount of change in data.

```
#Before Outliers treatment
fig,axes = plt.subplots(nrows = 2,ncols = 3,figsize=(10,15))
sns.histplot(data = df,x='cutoff_factor',kde=True,ax = axes[0,0])
sns.histplot(data = df,x='actual_time',kde=True,ax = axes[1,0])
sns.histplot(data = df,x='segment_actual_time',kde=True,ax = axes[0,1])
sns.histplot(data = df,x='segment_osrm_time',kde=True,ax = axes[1,1])
sns.histplot(data = df,x='new_col',kde=True,ax = axes[0,2])
sns.histplot(x = df['start_scan_to_end_scan'],kde=True,ax=axes[1,2])
plt.show()
```

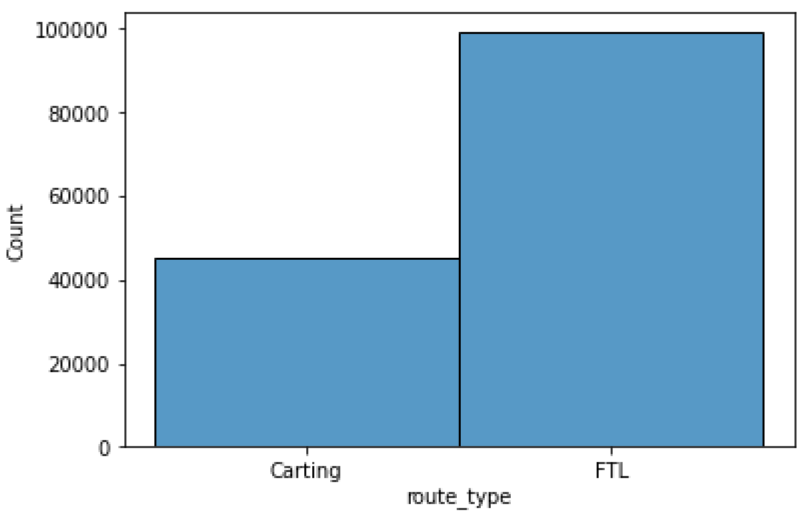


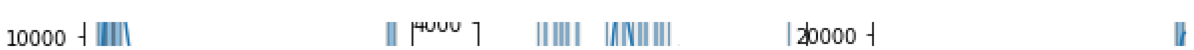
```
#After Outliers treatment
fig,axes = plt.subplots(nrows = 2,ncols = 3,figsize=(10,15))
sns.histplot(data = new_df_cap,x='cutoff_factor',kde=True,ax = axes[0,0])
sns.histplot(data = new_df_cap,x='actual_time',kde=True,ax = axes[1,0])
sns.histplot(data = new_df_cap,x='segment_actual_time',kde=True,ax = axes[0,1])
sns.histplot(data = new_df_cap,x='segment_osrm_time',kde=True,ax = axes[1,1])
sns.histplot(data = new_df_cap,x='new_col',kde=True,ax = axes[0,2])
sns.histplot(x = new_df_cap['start_scan_to_end_scan'],kde=True,ax = axes[1,2])
plt.show()
```



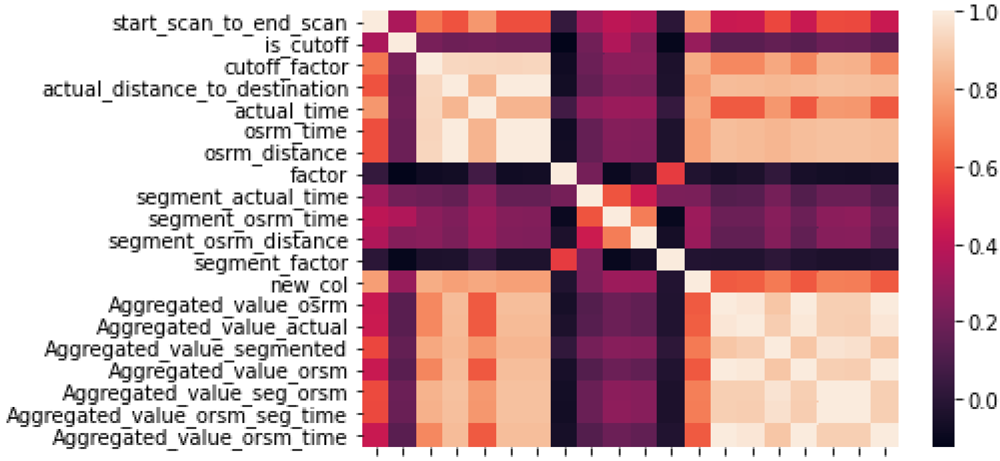
We can observe that there are significant amount of change in data after Handle of Outliers.

```
sns.histplot(data = df,x='route_type')
plt.show()
```






```
sns.heatmap(new_df_cap.corr())
plt.show()
```




```
new_df_cap.head(2)
```

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	so
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IN
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IN

2 rows × 33 columns

As we can see from above plots that Start_scan_to_end_scan and the column with the difference between timing in OD tends to have same distribution i.e right skewed distribution.

Does it have any dependency?

Let us check.

Null Hypothesis : Both columns are Independent of each other.

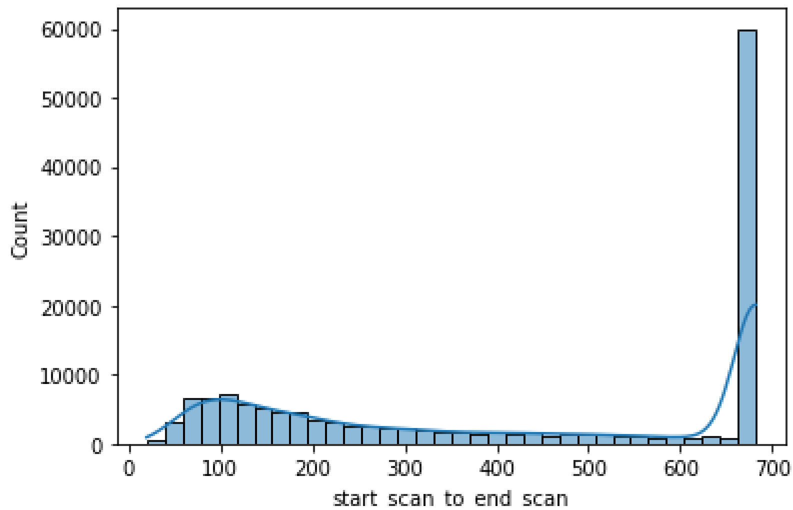
Alternate Hypothesis : Both columns are dependent.

Significance Value: 0.05

Test : Chi2 Test

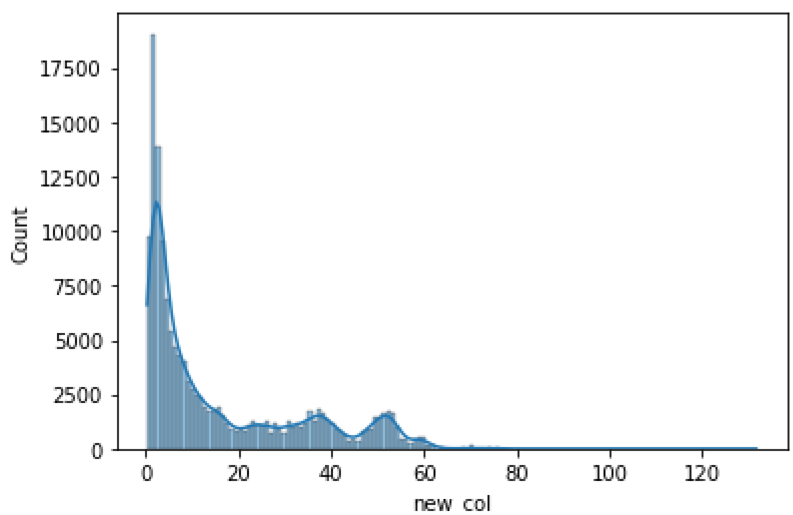
Hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value.

```
sns.histplot(x=new_df_cap['start_scan_to_end_scan'],kde=True)
plt.show()
```




```
sns.histplot(x=new_df_cap['new_col'],kde=True)
```

    <matplotlib.axes._subplots.AxesSubplot at 0x7ff79435f6d0>




```
data3 = new_df_cap['start_scan_to_end_scan'].sample(frac=0.8)
data4 =new_df_cap['new_col'].sample(frac = 0.8)
stat, p, dof, expected = chi2_contingency(data3,data4)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
  print('Probably the same distribution')
else:
  print('Probably different distributions')
```

```
stat=0.000, p=1.000
Probably the same distribution
```

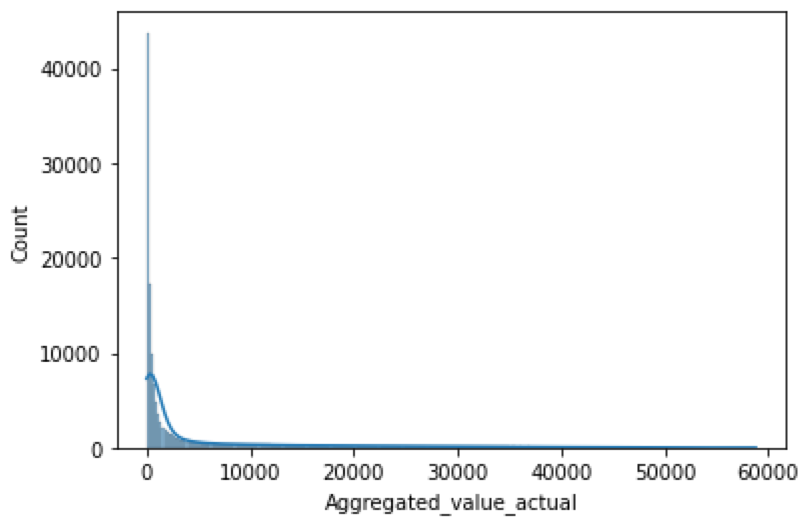
```
new_df_cap.drop(['Aggregated_value_osrm',
       'Aggregated_value_actual', 'Aggregated_value_segmented',
       'Aggregated_value_orsm', 'Aggregated_value_seg_orsm',
       'Aggregated_value_orsm_seg_time', 'Aggregated_value_orsm_time'],axis=1,inplace=True)
```

Hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value.

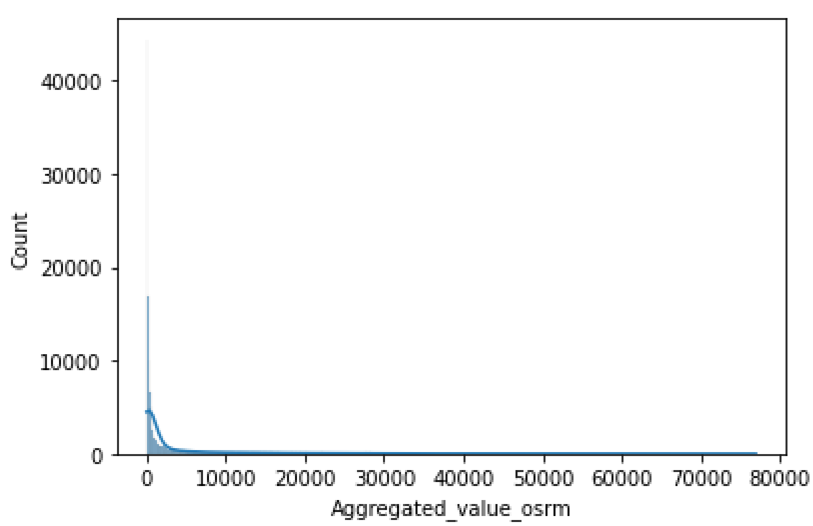
```
new_df_cap['Aggregated_value_osrm'] = new_df_cap.groupby(['trip_uuid'])['osrm_time'].cumsum()
```

```
new_df_cap['Aggregated_value_actual'] = new_df_cap.groupby(['trip_uuid'])['actual_time'].cums
```

```
sns.histplot(data = new_df_cap, x='Aggregated_value_actual',kde=True)
plt.show()
```




```
sns.histplot(data = new_df_cap, x='Aggregated_value_osrm',kde=True)
plt.show()
```



As the distribution of both the columns seems to be have same, lets see be doing Hypothesis Testing on both of these columns with sample size of 80% of the data.

HO : Both columns are Dependent.

Ha : Both columns are Independent.

Significance Value = 0.05

Test : Pearson Test

```
data1 = new_df_cap['Aggregated_value_osrm'].sample(frac = 0.8)
data2 = new_df_cap['Aggregated_value_actual'].sample(frac = 0.8)
stat, p = pearsonr(data1,data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
    print('Probably independent')
else:
    print('Probably dependent')
```

```
stat=0.001, p=0.709
Probably independent
```

Try the data with Chi2 Contingency test:

```
data1 = new_df_cap['Aggregated_value_osrm'].sample(frac = 0.8)
data2 = new_df_cap['Aggregated_value_actual'].sample(frac = 0.8)

stat, p = spearmanr(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
    print('Probably independent')
else:
    print('Probably dependent')
```

```
stat=-0.005, p=0.068
Probably independent
```

From above Hypothesis Tests we can conclude that we can reject our Null Hypothesis and we can conclude that both columns data are Independent.

Hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value.

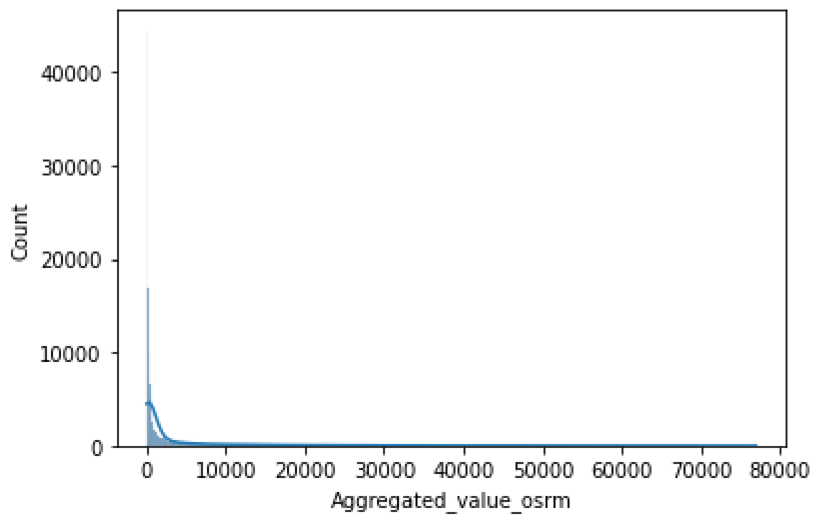
Ho : Both Columns are Similar

Ha : Both Columns are Independent.

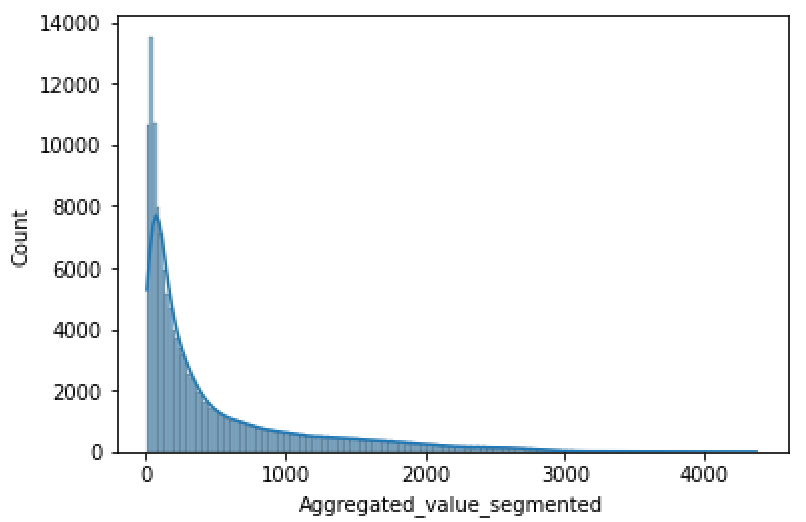
Test : Pearson

```
new_df_cap['Aggregated_value_segmented'] = new_df_cap.groupby(['trip_uuid'])['segment_actual_
```

```
sns.histplot(data = new_df_cap, x='Aggregated_value_osrm',kde=True)
plt.show()
```




```
sns.histplot(data = new_df_cap, x='Aggregated_value_segmented',kde=True)
plt.show()
```




```
data5 = new_df_cap['Aggregated_value_osrm'].sample(frac=0.8)
data6 = new_df_cap['Aggregated_value_segmented'].sample(frac=0.8)
stat, p = pearsonr(data5,data6)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
    print('Probably independent')
else:
    print('Probably dependent')
```

```
stat=0.002, p=0.563
Probably independent
```

Test Statistics : Spearman

```
stat, p = spearmanr(data5,data6)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
    print('Probably independent')
else:
    print('Probably dependent')
```

```
stat=-0.003, p=0.384
Probably independent
```

As we can see that in both Test statistics, we can reject our Null HHypothesis.

Do hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value.

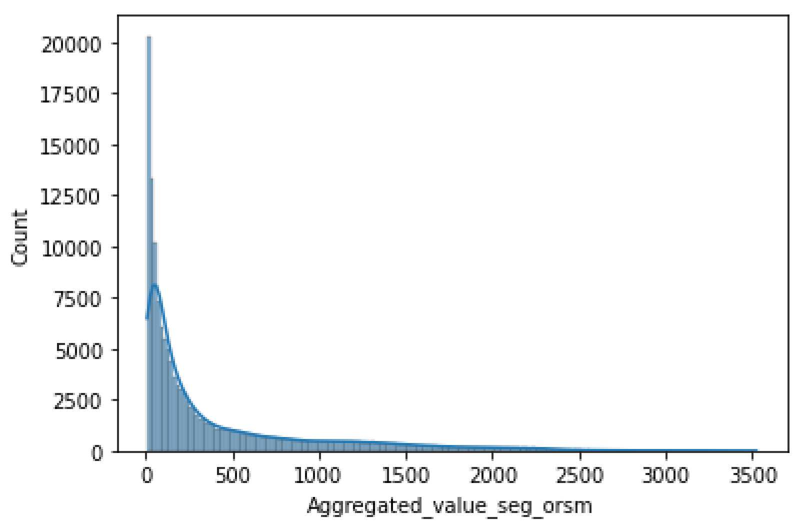
Ho : Both Columns are Similar

Ha : Both Columns are Independent.

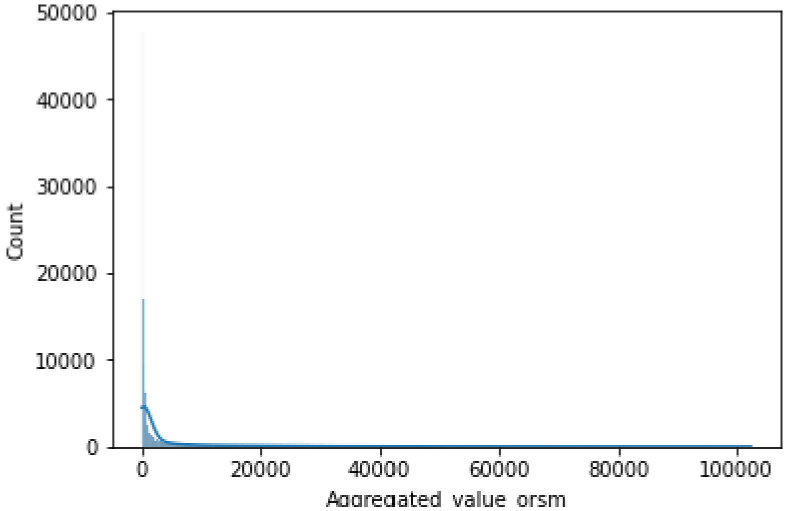
Test : Pearson

```
new_df_cap['Aggregated_value_orsm'] = new_df_cap.groupby(['trip_uuid'])['osrm_distance'].cums
new_df_cap['Aggregated_value_seg_orsm'] = new_df_cap.groupby(['trip_uuid'])['segment_osrm_dis
```

```
sns.histplot(data=new_df_cap,x='Aggregated_value_seg_orsm',kde=True)
plt.show()
```




```
sns.histplot(data=new_df_cap,x='Aggregated_value_orsm',kde=True)
plt.show()
```




```
data7 = new_df_cap['Aggregated_value_orsm'].sample(frac=0.8)
data8 = new_df_cap['Aggregated_value_seg_orsm'].sample(frac=0.8)
stat, p = pearsonr(data7,data8)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
  print('Probably independent')
else:
  print('Probably dependent')
```

```
stat=0.001, p=0.771
Probably independent
```

Do hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value.

Ho : Both Columns are Similar

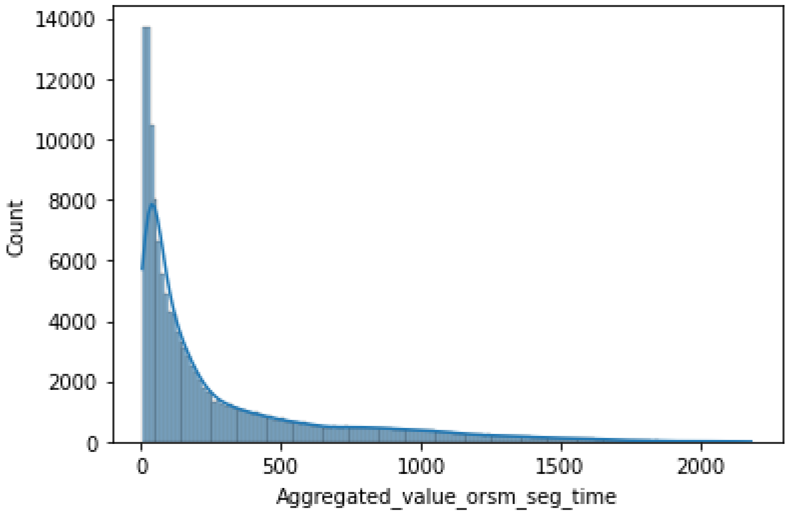
Ha : Both Columns are Independent.

Test : Pearson

```
new_df_cap['Aggregated_value_orsm_seg_time'] = new_df_cap.groupby(['trip_uuid'])['segment_osr
new_df_cap['Aggregated_value_orsm_time'] = new_df_cap.groupby(['trip_uuid'])['osrm_time'].cum
```

```
sns.histplot(data = new_df_cap, x='Aggregated_value_orsm_time',kde=True)
plt.show()
```


```
sns.histplot(data = new_df_cap, x='Aggregated_value_orsm_seg_time',kde=True)
plt.show()
```




```
data9 = new_df_cap['Aggregated_value_orsm_seg_time'].sample(frac=0.8)
data10 = new_df_cap['Aggregated_value_orsm_time'].sample(frac=0.8)
stat, p = pearsonr(data9,data10)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
    print('Probably independent')
else:
    print('Probably dependent')
```

```
stat=0.003, p=0.296
Probably independent
```

Test Statistics : Spearman

```
from scipy.stats import spearmanr
stat, p = spearmanr(data9, data10)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
  print('Probably independent')
else:
  print('Probably dependent')
```

```
stat=0.007, p=0.116
Probably independent
```

0s completed at 6:20 PM